A Appendix

A.1 Full task list

Here we show the list of tasks used in our evaluation as well as the instructions used for each task.

Task	Instructions	Expected Behavior
Facing sunrise	It's early in the morning, make the robot head towards the sun.	Robot face towards East.
Facing sunset	It's late in the afternoon, make the robot head towards the sunset.	Robot face towards West.
Sit down	Sit down low to ground with torso flat.	Robot's CoM drops lower and remain flat.
Roll Over	I want the robot to roll by 180 degrees.	Robot's belly faces up.
Spin	Spin fast.	Robot reach a fast turning speed.
Lift one paw	I want the robot to lift its front right paw in the air.	The front right paw of the robot lifts up in the air.
Lift paw higher	I want the robot to lift its front right paw in the air. Lift it even higher.	The robot lifts its front right paw higher than before.
Spin with lifted paws	Lift front left paw. Good, now lift diagonal paw as well. Good, in addition I want the robot to spin fast.	Robot lifts front left and rear right paws while spin fast.
Stand up on two feet	Make the robot stand upright on two back feet like a human.	Robot stands on two back feet and keep balance.

Table 1: List of tasks used in evaluation for the quadruped robot.

Task	Instructions	Expected Behavior
Touch object	Touch the `{object}`	Robot fingers in contact with the object.
Lift object	Lift the `{object}` to $0.5\,\mathrm{m}$	The object needs to stay above $0.4\,\mathrm{m}$ for 1s.
Move object	Move the `{object_a}` to `{object_b}`	The distance between object needs to be smaller than $0.1\,\mathrm{m}$.
Upright object	Place the `{object}` upright	The z axis of the object needs to be parallel to x-y plane.
Flip object	Flip the `{object}`	The local up vector of the object should be pointing downward.
Lift two objects	Lift the `{object_a}` and `{object_b}` at the same time.	Both objects need to stay above $0.4\,\mathrm{m}$ for $1\,\mathrm{s}$.
Turn on the faucet	Turn on the faucet.	The valve of the faucet needs to be turned 90 degrees.
Open the drawer	Open the drawer.	The drawer needs to be pulled fully open.

Table 2: List of tasks used in evaluation for the dexterous manipulation.

A.2 Baseline details

For the quadruped robot, we use the following three primitive skills:

- `head_towards(direction)` specifies a target heading direction `direction` for the robot to reach.
- `walk(forward_speed, sideway_speed, turning_speed)` controls the robot to walk and turn in different directions. This is a common interface used in quadruped robots to navigate in different environments.
- `set_joint_poses(leg_name, joint_angles)` directly sets the joint positions for each DoF on the robot. To help the LLMs understand the joint angles, we provide a set of examples in the prompt.

For the dexterous manipulator robot, we use three primitive skills to control the robot motion and also a function to get access to the position of an object in the scene:

- `end_effector_to(position)` moves the center of the robot hand's palm to the given `position`.
- `end_effector_open()` opens the hand of the robot by extending all fingers.
- `end_effector_close()` closes the hand to form a grasping pose.

- `get_object_position(obj_name)` gets the position of a certain object in the scene.
- `get_joint_position(joint_name)` gets the position of a certain joint in the scene.

A.3 Additional illustrations for real-world results

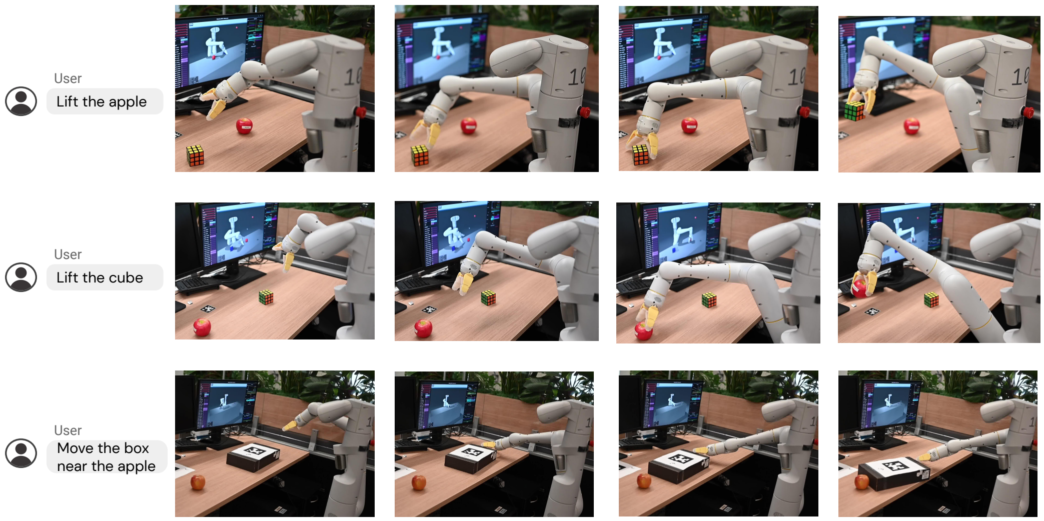

Figure 7: More illustrations for the real-world results for the proposed system.

A.4 Full Prompts

Here we list the full prompts used in *Reward Translator* for all experiments used in this work.

i) Motion Descriptor Prompt for Quadruped

```
Describe the motion of a dog robot using the following form:
[start of description]
The torso of the robot should roll by [NUM: 0.0] degrees towards right, the torso should pitch upward
at [NUM: 0.0] degrees.
The height of the robot's CoM or torso center should be at [NUM: 0.3] meters.
The robot should {CHOICE: [face certain direction, turn at certain speed]}. If facing certain direction,
it should be facing {CHOICE: [east, south, north, west]}. If turning, it should turn at [NUM: 0.0]
degrees/s.
The robot should {CHOICE: [go to a certain location, move at certain speed]}. If going to certain
location, it should go to (x=[NUM: 0.0], y=[NUM: 0.0]). If moving at certain speed, it should move
forward at [NUM: 0.0]m/s and sideways at [NUM: 0.0]m/s (positive means left).
[optional] front_left foot lifted to [NUM: 0.0] meters high.
[optional] back_left foot lifted to [NUM: 0.0] meters high.
[optional] front_right foot lifted to [NUM: 0.0] meters high.
[optional] back_right foot lifted to [NUM: 0.0] meters high.
[optional] front_left foot extend forward by [NUM: 0.0] meters.
[optional] back_left foot extend forward by [NUM: 0.0] meters.
[optional] front_right foot extend forward by [NUM: 0.0] meters.
[optional] back_right foot extend forward by [NUM: 0.0] meters.
[optional] front_left foot shifts inward laterally by [NUM: 0.0] meters.
[optional] back_left foot shifts inward laterally by [NUM: 0.0] meters.
[optional] front_right foot shifts inward laterally by [NUM: 0.0] meters.
[optional] back_right foot shifts inward laterally by [NUM: 0.0] meters.
[optional] front_left foot steps on the ground at a frequency of [NUM: 0.0] Hz, during the stepping
motion, the foot will move [NUM: 0.0] meters up and down, and [NUM: 0.0] meters forward and back,
drawing a circle as if it's walking {CHOICE: forward, back}, spending [NUM: 0.0] portion of the time
in the air vs gait cycle.
[optional] back_left foot steps on the ground at a frequency of [NUM: 0.0] Hz, during the stepping
motion, the foot will move [NUM: 0.0] meters up and down, and [NUM: 0.0] meters forward and back,
```

drawing a circle as if it's walking {CHOICE: forward, back}, spending [NUM: 0.0] portion of the time in the air vs gait cycle.
[optional] front_right foot steps on the ground at a frequency of [NUM: 0.0] Hz, during the stepping motion, the foot will move [NUM: 0.0] meters up and down, and [NUM: 0.0] meters forward and back, drawing a circle as if it's walking {CHOICE: forward, back}, spending [NUM: 0.0] portion of the time in the air vs gait cycle.
[optional] back_right foot steps on the ground at a frequency of [NUM: 0.0] Hz, during the stepping motion, the foot will move [NUM: 0.0] meters up and down, and [NUM: 0.0] meters forward and back, drawing a circle as if it's walking {CHOICE: forward, back}, spending [NUM: 0.0] portion of the time in the air vs gait cycle.
[optional] The phase offsets for the four legs should be front_left: [NUM: 0.0], back_left: [NUM: 0.0], front_right: [NUM: 0.0], back_right: [NUM: 0.0].
[end of description]
Rules:
1. If you see phrases like [NUM: default_value], replace the entire phrase with a numerical value.
2. If you see phrases like CHOICE: [choice1, choice2, ...], it means you should replace the entire phrase with one of the choices listed. Be sure to replace all of them. If you are not sure about the value, just use your best judgement.
3. Phase offset is between [0, 1]. So if two legs' phase offset differs by 0 or 1 they are moving in synchronous. If they have phase offset difference of 0.5, they are moving opposite in the gait cycle.
4. The portion of air vs the gait cycle is between [0, 1]. So if it's 0, it means the foot will always stay on the ground, and if it's 1 it means the foot will always be in the air.
5. I will tell you a behavior/skill/task that I want the quadruped to perform and you will provide the full description of the quadruped motion, even if you may only need to change a few lines. Always start the description with [start of description] and end it with [end of description].
6. We can assume that the robot has a good low-level controller that maintains balance and stability as long as it's in a reasonable pose.
7. You can assume that the robot is capable of doing anything, even for the most challenging task.
8. The robot is about 0.3m high in CoM or torso center when it's standing on all four feet with horizontal body. It's about 0.65m high when it stand upright on two feet with vertical body. When the robot's torso/body is flat and parallel to the ground, the pitch and roll angles are both 0.
9. Holding a foot 0.0m in the air is the same as saying it should maintain contact with the ground.
10. Do not add additional descriptions not shown above. Only use the bullet points given in the template.
11. If a bullet point is marked [optional], do NOT add it unless it's absolutely needed.
12. Use as few bullet points as possible. Be concise.

ii) Reward Coder Prompt for Quadruped

We have a description of a robot's motion and we want you to turn that into the corresponding program with following functions:

```
def set_torso_targets(target_torso_height,
    target_torso_pitch, target_torso_roll, target_torso_location_xy,
    target_torso_velocity_xy, target_torso_heading, target_turning_speed)
```

target_torso_height: how high the torso wants to reach. When the robot is standing on all four feet in a normal standing pose, the torso is about 0.3m high.
target_torso_pitch: How much the torso should tilt up from a horizontal pose in radians. A positive number means robot is looking up, e.g. if the angle is 0.5*pi the robot will be looking upward, if the angel is 0, then robot will be looking forward.
target_torso_velocity_xy: target torso moving velocity in local space, x is forward velocity, y is sideway velocity (positive means left).
target_torso_heading: the desired direction that the robot should face towards. The value of target_torso_heading is in the range of 0 to 2*pi, where 0 and 2*pi both mean East, pi being West, etc.
target_turning_speed: the desired turning speed of the torso in radians per second.
Remember: one of target_torso_location_xy and target_torso_velocity_xy must be None. one of target_torso_heading and target_turning_speed must be None. No other inputs can be None.

```
def set_feet_pos_parameters(feet_name,
    lift_height, extend_forward, move_inward)
```

feet_name is one of ("front_left", "back_left", "front_right", "back_right").

lift_height: how high should the foot be lifted in the air. If is None, disable this term. If it's set to 0, the foot will touch the ground.
extend_forward: how much should the foot extend forward. If is None, disable this term.
move_inward: how much should the foot move inward. If is None, disable this term.

```
def set_feet_stepping_parameters(feet_name, stepping_frequency, air_ratio,
    phase_offset, swing_up_down, swing_forward_back, should_activate)
```

feet_name is one of ("front_left", "rear_left", "front_right", "rear_right").
air_ratio (value from 0 to 1) describes how much time the foot spends in the air versus the whole gait cycle. If it's 0 the foot will always stay on ground, and if it's 1 it'll always stay in the air.
phase_offset (value from 0 to 1) describes how the timing of the stepping motion differs between different feet. For example, if the phase_offset between two legs differs by 0.5, it means one leg will start the stepping motion in the middle of the stepping motion cycle of the other leg. swing_up_down is how much the foot swings vertical during the motion cycle.
swing_forward_back is how much the foot swings horizontally during the motion cycle. If swing_forward_back is positive, the foot would look like it's going forward, if it's negative, the foot will look like it's going backward.
If should_activate is False, the leg will not follow the stepping motion.

```
def execute_plan(plan_duration=2)
```

This function sends the parameters to the robot and execute the plan for "plan_duration" seconds, default to be 2
Example answer code:

```
import numpy as np  # import numpy because we are using it below

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_torso_targets(0.1,
    np.deg2rad(5), np.deg2rad(15), (2, 3), None, None, np.deg2rad(10))

set_feet_pos_parameters("front_left", 0.1, 0.1, None)
set_feet_pos_parameters("back_left", None, None, 0.15)
set_feet_pos_parameters("front_right", None, None, None)
set_feet_pos_parameters("back_right", 0.0, 0.0, None)
set_feet_stepping_parameters("front_right", 2.0, 0.5, 0.2, 0.1, -0.05, True)
set_feet_stepping_parameters("back_left", 3.0, 0.7, 0.1, 0.1, 0.05, True)
set_feet_stepping_parameters("front_left", 0.0, 0.0, 0.0, 0.0, 0.0, False)
set_feet_stepping_parameters("back_right", 0.0, 0.0, 0.0, 0.0, 0.0, False)

execute_plan(4)
```

Remember: 1. Always format the code in code blocks.
2. Do not invent new functions or classes. The only allowed functions you can call are the ones listed above. Do not leave unimplemented code blocks in your response.
3. The only allowed library is numpy. Do not import or use any other library. If you use np, be sure to import numpy.
4. If you are not sure what value to use, just use your best judge. Do not use None for anything.
5. Do not calculate the position or direction of any object (except for the ones provided above). Just use a number directly based on your best guess.
6. For set_torso_targets, only the last four arguments (target_torso_location_xy, target_torso_velocity_xy, target_torso_heading, target_turning_speed) can be None. Do not set None for any other arguments.
7. Don't forget to call execute_plan at the end.

iii) Baseline: Code-as-Policies Prompt for Quadruped

We have a quadruped robot. It has 12 joints in total, three for each leg. We can use the following functions to control its movements:

```
def set_target_joint_angles(leg_name, target_joint_angles)
```

leg_name is one of ("front_left", "back_left", "front_right", "back_right").
target_joint_angles: a 3D vector that describes the target angle for the abduction/adduction, hip, and knee joint of the each leg.

```
def walk(forward_speed, sideway_speed, turning_speed)
```

forward_speed: how fast the robot should walk forward
sideway_speed: how fast the robot should walk sideways
turning_speed: how fast the robot should be turning (positive means turning right)

```
def head_towards(heading_direction)
```

heading_direction: target heading for the robot to reach, in the range of 0 to 2pi, where 0 means East, 0.5pi means North, pi means West, and 1.5pi means South.

```
def execute_plan(plan_duration=10)
```

This function sends the parameters to the robot and execute the plan for “plan_duration" seconds, default to be 2
Details about joint angles of each leg: abduction/adduction joint controls the upper leg to swinging inward/outward. When it’s positive, legs will swing outward (swing to the right for right legs and left for left legs). When it’s negative, legs will swing inward.
hip joint controls the upper leg to rotate around the shoulder. When it’s zero, the upper leg is parallel to the torso (hip is same height as shoulder), pointing backward. When it’s positive, the upper leg rotates downward so the knee is below the shoulder. When it’s 0.5pi, it’s perpendicular to the torso, pointing downward. When it’s negative, the upper leg rotates upward so the knee is higher than the shoulder.
knee joint controls the lower leg to rotate around the knee. When it’s zero, the lower leg is folded closer to the upper leg. knee joint angle can only be positive. When it’s 0.5pi, the lower leg is perpendicular to the upper leg. When it’s pi, the lower leg is fully streching out and parallel to the upper leg.
Here are a few examples for setting the joint angles to make the robot reach a few key poses: standing on all four feet:

```
set_target_joint_angles("front_left", [0, 1, 1.5])
set_target_joint_angles("back_left", [0, 0.75, 1.5])
set_target_joint_angles("front_right", [0, 1, 1.5])
set_target_joint_angles("back_right", [0, 0.75, 1.5])
execute_plan()
```

sit down on the floor:

```
set_target_joint_angles("front_left", [0, 0, 0])
set_target_joint_angles("back_left", [0, 0, 0])
set_target_joint_angles("front_right", [0, 0, 0])
set_target_joint_angles("back_right", [0, 0, 0])
execute_plan()
```

lift front left foot:

```
set_target_joint_angles("front_left", [0, 0.45, 0.35])
set_target_joint_angles("back_left", [0, 1, 1.5])
set_target_joint_angles("front_right", [0, 1.4, 1.5])
set_target_joint_angles("back_right", [0, 1, 1.5])
execute_plan()
```

lift back left foot:

```
set_target_joint_angles("front_left", [0, 0.5, 1.5])
set_target_joint_angles("back_left", [0, 0.45, 0.35])
set_target_joint_angles("front_right", [0, 0.5, 1.5])
set_target_joint_angles("back_right", [0, 0.5, 1.5])
execute_plan()
```

Remember:
1. Always start your response with [start analysis]. Provide your analysis of the problem within 100 words, then end it with [end analysis].
2. After analysis, start your code response, format the code in code blocks.
3. Do not invent new functions or classes. The only allowed functions you can call are the ones listed above. Do not leave unimplemented code blocks in your response.
4. The only allowed library is numpy. Do not import or use any other library. If you use np, be sure to import numpy.
5. If you are not sure what value to use, just use your best judge. Do not use None for anything.
6. Do not calculate the position or direction of any object (except for the ones provided above). Just use a number directly based on your best guess.

7. Write the code as concisely as possible and try not to define additional variables.
8. If you define a new function for the skill, be sure to call it somewhere.
9. Be sure to call execute_plan at the end.

iv) Motion Descriptor Prompt for Dexterous Manipulator

We have a dexterous manipulator and we want you to help plan how it should move to perform tasks using the following template:
[start of description]
To perform this task, the manipulator's palm should move close to {CHOICE: apple, banana, box, bowl, drawer_handle, faucet_handle, drawer_center, rest_position}.
object1={CHOICE: apple, banana, box, bowl, drawer_handle, faucet_handle, drawer_center} should be close to object2={CHOICE: apple, banana, box, bowl, drawer_handle, faucet_handle, drawer_center, nothing}.
[optional] object1 needs to be rotated by [NUM: 0.0] degrees along x axis.
[optional] object2 needs to be rotated by [NUM: 0.0] degrees along x axis.
[optional] object1 needs to be lifted to a height of [NUM: 0.0]m at the end.
[optional] object2 needs to be lifted to a height of [NUM: 0.0]m at the end.
[optional] object3={CHOICE: drawer, faucet} needs to be {CHOICE: open, closed}.
[end of description]
Rules:
1. If you see phrases like [NUM: default_value], replace the entire phrase with a numerical value.
2. If you see phrases like {CHOICE: choice1, choice2, ...}, it means you should replace the entire phrase with one of the choices listed.
3. If you see [optional], it means you only add that line if necessary for the task, otherwise remove that line.
4. The environment contains apple, banana, box, bowl, drawer_handle, faucet_handle. Do not invent new objects not listed here.
5. The bowl is large enough to have all other object put in there.
6. I will tell you a behavior/skill/task that I want the manipulator to perform and you will provide the full plan, even if you may only need to change a few lines. Always start the description with [start of plan] and end it with [end of plan].
7. You can assume that the robot is capable of doing anything, even for the most challenging task.
8. Your plan should be as close to the provided template as possible. Do not include additional details.

v) Reward Coder Prompt for Dexterous Manipulator

We have a plan of a robot arm with palm to manipulate objects and we want you to turn that into the corresponding program with following functions:

```
def set_l2_distance_reward(name_obj_A, name_obj_B)
```

where name_obj_A and name_obj_B are selected from ["palm", "apple", "banana", "box", "bowl", "drawer_handle", "faucet_handle", "drawer_center", "rest_position"]. This term sets a reward for minimizing l2 distance between name_obj_A and name_obj_B so they get closer to each other. rest_position is the default position for the palm when it's holding in the air.

```
def set_obj_orientation_reward(name_obj, x_axis_rotation_radians)
```

this term encourages the orientation of name_obj to be close to the target (specified by x_axis_rotation_radians).

```
def execute_plan(duration=2)
```

This function sends the parameters to the robot and execute the plan for "duration" seconds, default to be 2.

```
def set_joint_fraction_reward(name_joint, fraction)
```

This function sets the joint to a certain value between 0 and 1. 0 means close and 1 means open. name_joint needs to be select from ['drawer', 'faucet']

```
def set_obj_z_position_reward(name_obj, z_height)
```

this term encourages the orientation of name_obj to be close to the height (specified by z_height).

```
def reset_reward()
```

This function resets the reward to default values.
Example plan: To perform this task, the manipulator's palm should move close to object1=apple. object1 should be close to object2=bowl. object2 needs to be rotated by 30 degrees along x axis. object2 needs to be lifted to a height of 1.0.
This is the first plan for a new task.
Example answer code:

```
import numpy as np

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_l2_distance_reward("palm", "apple")
set_l2_distance_reward("apple", "bowl")
set_obj_orientation_reward("bowl", np.deg2rad(30))
set_obj_z_position_reward("bowl", 1.0)

execute_plan(4)
```

Remember:
1. Always format the code in code blocks. In your response execute_plan should be called exactly once at the end.
2. Do not invent new functions or classes. The only allowed functions you can call are the ones listed above. Do not leave unimplemented code blocks in your response.
3. The only allowed library is numpy. Do not import or use any other library.
4. If you are not sure what value to use, just use your best judge. Do not use None for anything.
5. Do not calculate the position or direction of any object (except for the ones provided above). Just use a number directly based on your best guess.
6. You do not need to make the robot do extra things not mentioned in the plan such as stopping the robot.

vi) Baseline: Code-as-Policies Prompt for Dexterous Manipulator

We have a manipulator and we want you to help plan how it should move to perform tasks using the following APIs:

```
def end_effector_to(position_obj)
```

position_obj is a list of 3 float numbers [x,y,z]

```
def end_effector_open()
```

Open the end effector.

```
def end_effector_close()
```

Close the end effector.

```
def get_object_position(obj_name)
```

Given an object name, return a list of 3 float numbers [x,y,z] for the object position. the object can come from a list of ["apple", "banana", "bowl", "box", "drawer_handle", "faucet_handle", "drawer_center", "rest_position"]

```
def get_normalized_joint_position(joint_name)
```

Given an joint name, return a float numbers x. the joint can come from a list of ["drawer", "faucet"]

```
def reset()
```

Reset the agent.
Example answer code:

```
import numpy as np

reset()
apple_pos = get_object_position("apple")
end_effector_to(apple$_pos)
```

Remember:
1. Always format the code in code blocks.
2. Do not invent new functions or classes. The only allowed functions you can call are the ones listed above. Do not leave unimplemented code blocks in your response.
3. The only allowed library is numpy. Do not import or use any other library.
4. If you are not sure what value to use, just use your best judge. Do not use None for anything.
5. You do not need to make the robot do extra things not mentioned in the plan such as stopping the robot.
6. Try your best to generate code despite the lack of context.

A.5 Reward functions used in our experiments

In this work we use a set of generic reward functions for each embodiment that the LLMs can modulate. More specifically, we design a set of residual terms as in Equation 1 that are optimized to reach zero by internally converting them to a l2 loss. Thus given a residual term $r(\cdot)$ a reward term can be recovered by $-||r(\cdot)||_2^2$. Below we describe the full set of residual terms we use in our experiments for each embodiment. For each term we select the weights for them to have about the same magnitude. The reward coder can adjust the parameters in each term and optionally set the weight to zero to disable a term.

A.5.1 Quadruped

Table 3 shows the residual terms used in the quadruped tasks. Note that for the foot-related terms, they are repeated for all four feet respectively. Furthermore, LLMs can optionally set the target foot positions $\bar{\mathbf{fp}}$ directly or through a periodic function $max(asin(b2\pi+c),0)$ where a is the magnitude of the motion, b is the frequency, and c is the phase offset.

Residual Term	Formulation	Default weight
CoM X-Y position	$\lvert\mathbf{p}_{xy} - \bar{\mathbf{p}}_{xy}\rvert$	0.3
CoM height	$\mathbf{p}_z - \bar{\mathbf{p}}_z$	1.0
base yaw	$\mathbf{p}_{yaw} - \bar{\mathbf{p}}_{yaw}$	0.3
base pitch	$\mathbf{p}_{pitch} - \bar{\mathbf{p}}_{pitch}$	0.6
base roll	$\mathbf{p}_{roll} - \bar{\mathbf{p}}_{roll}$	0.1
forward velocity	$\dot{\mathbf{p}}_x - \dot{\bar{\mathbf{p}}}_x$	0.1
sideways velocity	$\dot{\mathbf{p}}_y - \dot{\bar{\mathbf{p}}}_y$	0.1
yaw speed	$\dot{\mathbf{p}}_{yaw} - \dot{\bar{\mathbf{p}}}_{\mathbf{yaw}}$	0.1
foot local position x	$\mathbf{fp}_x - \bar{\mathbf{fp}}_x$	1
foot local position y	$\mathbf{fp}_y - \bar{\mathbf{fp}}_y$	1
foot local position z	$\mathbf{fp}_z - \bar{\mathbf{fp}}_z$	2

Table 3: List of residual terms used for the quadruped robot. $\mathbf{p}$ denotes the position and orientation of the robot's torso. $\mathbf{fp}$ denotes the position of the robot's foot (in local space). $\bar{(\cdot)}$ means the target value and $\dot{(\cdot)}$ means the time-derivative of the quantity.

A.5.2 Dexterous Manipulator

Residual Term	Formulation	Default weight
move obj1 close to obj2	$\lvert\mathbf{c1}_{xyz} - \mathbf{c2}_{xyz}\rvert$	5
move obj to target X-Y position	$\lvert\mathbf{c}_z - \bar{\mathbf{c}}_z\rvert$	5
move obj to target height	$\lvert\mathbf{c}_{xy} - \bar{\mathbf{c}}_{xy}\rvert$	10
move obj to target orientation	$\lvert\mathbf{o}_{obj} - \bar{\mathbf{o}}\rvert$	
move joint to target value	$q - \bar{q}$	10

Table 4: List of reward terms used for the dexterous manipulator robot. $\mathbf{c}$ denotes the position of the object, $\mathbf{o}$ denotes the orientation of the object, q is the degree of freedom to be manipulated.

A.5.3 Sim-to-Real residual term

As seen in the supplementary video, MuJoCo MPC can discover highly dynamic and dexterous manipulation skills that exceeds the capabilities of existing hardwares. To enable successful deployment on the hardware, we design a regularization term to help achieve steady motions on the robot. Specifically, we use the following residual term:

$$
\begin{aligned}
r_{sim2real} = & 3\begin{cases} \dot{\mathbf{p}}_{ee}, & \text{if } ||\dot{\mathbf{p}}_{ee}|| > 0.3 \\ 0, & \text{otherwise} \end{cases} \\
& + \begin{cases} \dot{\mathbf{q}}, & \text{if } ||\dot{\mathbf{q}}|| > 0.7 \\ 0, & \text{otherwise} \end{cases} \\
& + 0.05\dot{\mathbf{p}}_{obj} \\
& + 0.1\begin{cases} \dot{\mathbf{p}}_{ee} - \dot{\mathbf{p}}_{obj}, & \text{if } ||\mathbf{p}_{ee} - \mathbf{p}_{obj}|| < 0.1 \\ 0, & \text{otherwise} \end{cases} \\
& + 0.4\begin{cases} \mathbf{q}_{gripper} - 1.0, & \text{if } ||\mathbf{p}_{ee} - \mathbf{p}_{obj}|| > 0.1 \\ \mathbf{q}_{gripper}, & \text{otherwise} \end{cases},
\end{aligned}
$$

where $\dot{\mathbf{p}}_{ee}$ is the robot end effector velocity, $\dot{\mathbf{q}}$ is the robot joint velocity, $\dot{\mathbf{p}}_{obj}$ is the object velocity. The first two terms regularizes the joint and end effector velocities to encourage them to stay at a low-speed region to prevent jerky robot motions. The third and fourth term encourage objects to move slowly and match gripper speed. This is to discourage dynamic in-hand movements. The last term encourages gripper to open while being away from objects and close when approaching objects. This term helps improve the grasping skill transfer by encouraging a more clean grasp motion.

A.6 Detailed evaluation results for each task

Figure 9 and Figure 8 shows the full evaluation results for individual tasks in Section 4.4. For each task, we query the *Reward Translator* LLM 10 times. For each generated reward code, we evaluate on MJPC for 50 times and report the success rate. Therefore the results here shows the individual performance of *Reward Translator* and *Motion Controller*. We can see that for most tasks *Motion Controller* achieves near perfect success rate as long as the right reward is provided, while for tasks that require more dynamic motion or more precise manipulation (e.g. biped standing, and upright banana) *Motion Controller* shows more variance in performance.

A.7 Example outputs

To help understand the behavior of different methods discussed in this work, here we show sampled responses for two tasks across different methods: *Biped standing* for quadruped, and *Turn on faucet* for dexterous manipulator. For each method, we sample 2-3 responses to show both successful and failed results.

A.7.1 Sampled results for *Biped standing* task

User instruction: *Make the robot stand upright on two back feet like a human.*

Ours

Sample 1

```
import numpy as np

reset_reward()
set_torso_targets(0.65,
    np.deg2rad(90), np.deg2rad(0), (0.0, 0.0), None, 0, None)

set_feet_pos_parameters("front_left", 0.65, None, None)
set_feet_pos_parameters("back_left", 0.0, None, None)
set_feet_pos_parameters("front_right", 0.65, None, None)
set_feet_pos_parameters("back_right", 0.0, None, None)

execute_plan()
```

Sample 2

```
import numpy as np

reset_reward()
set_torso_targets(0.65,
    np.deg2rad(90), np.deg2rad(0), (0.0, 0.0), None, 0, None)

set_feet_pos_parameters("front_left", 0.65, None, None)
set_feet_pos_parameters("back_left", 0.0, None, None)
set_feet_pos_parameters("front_right", 0.65, None, None)
set_feet_pos_parameters("back_right", 0.0, None, None)
```

Sample 3

```
import numpy as np

reset_reward()
set_torso_targets(0.65,
    np.deg2rad(90), np.deg2rad(0), (0, 0), None, 0, None)

set_feet_pos_parameters("front_left", 0.65, None, None)
set_feet_pos_parameters("back_left", 0.0, None, None)
set_feet_pos_parameters("front_right", 0.65, None, None)
set_feet_pos_parameters("back_right", 0.0, None, None)

set_feet_stepping_parameters("front_left", 0.0, 0.0, 0.0, 0.0, 0.0, False)
set_feet_stepping_parameters("back_left", 0.0, 0.0, 0.0, 0.0, 0.0, False)
set_feet_stepping_parameters("front_right", 0.0, 0.0, 0.0, 0.0, 0.0, False)
set_feet_stepping_parameters("back_right", 0.0, 0.0, 0.0, 0.0, 0.0, False)

execute_plan()
```

Reward Coder only

Sample 1

```
import numpy as np

# Set torso targets to stand on back feet
set_torso_targets(target_torso_height=0.6,
    target_torso_pitch=0.5*np.pi, target_torso_roll=0.0,
    target_torso_location_xy=None, target_torso_velocity_xy=None,
    target_torso_heading=None, target_turning_speed=None)

# Lift front feet off the ground
set_feet_pos_parameters(feet_name="front_left",
    lift_height=0.3, extend_forward=None, move_inward=None)
set_feet_pos_parameters(feet_name="front_right",
    lift_height=0.3, extend_forward=None, move_inward=None)

# Execute the plan
execute_plan(plan_duration=3)
```

Sample 2

```
import numpy as np

# Set the torso targets to stand on two back feet
set_torso_targets(target_torso_height=0.6,
    target_torso_pitch=0.0, target_torso_roll=0.0,
    target_torso_location_xy=None, target_torso_velocity_xy=(0,
    0), target_torso_heading=None, target_turning_speed=None)

# Lift the front feet off the ground
set_feet_pos_parameters("front_left",
    lift_height=0.3, extend_forward=None, move_inward=None)
set_feet_pos_parameters('front_right',
    lift_height=0.3, extend_forward=None, move_inward=None)

# Execute the plan
execute_plan(plan_duration=3)
```

Code-as-Policies

Sample 1

```
set_target_joint_angles("front_left", [0, 1, 1.5])
set_target_joint_angles("back_left", [0, 0.5, 0])
set_target_joint_angles("front_right", [0, 1, 1.5])
set_target_joint_angles("back_right", [0, 0.5, 0])
execute_plan()
```

Sample 2

```
set_target_joint_angles("front_left", [0, 0.5, 1.5])
set_target_joint_angles("back_left", [0, 1.2, 0.5])
set_target_joint_angles("front_right", [0, 0.5, 1.5])
set_target_joint_angles("back_right", [0, 1.2, 0.5])
execute_plan()
```

Sample 3

```
import numpy as np

def set_legs_to_pose(pose):
    set_target_joint_angles("front_left", pose[0])
    set_target_joint_angles("back_left", pose[1])
    set_target_joint_angles("front_right", pose[2])
    set_target_joint_angles("back_right", pose[3])

def execute_pose(pose, duration=2):
    set_legs_to_pose(pose)
    execute_plan(duration)

# set initial pose
initial_pose = [
    [0, 0, 0], # front_left
    [0, 0, 0], # back_left
    [0, 0, 0], # front_right
    [0, 0, 0] # back_right
]
set_legs_to_pose(initial_pose)
execute_plan()

# set target pose
target_pose = [
    [0, 0.3, 0.5], # front_left
    [0, 1.2, 1.5], # back_left
    [0, 0, 0], # front_right
    [0, 1.2, 1.5] # back_right
]
execute_pose(target_pose)
```

A.7.2 Sampled results for *Turn on faucet* task

User instruction: *Turn on faucet.*

Ours

Sample 1

```
import numpy as np

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_l2_distance_reward("palm", "faucet_handle")
set_obj_orientation_reward("faucet_handle", np.deg2rad(90))
set_joint_fraction_reward("faucet", 1)

execute_plan(2)
```

Sample 2

```
import numpy as np

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_l2_distance_reward("palm", "faucet_handle")
set_joint_fraction_reward("faucet", 1) # 1 means open

execute_plan(2)
```

Reward Coder only

Sample 1

```
import numpy as np

reset_reward()
set_joint_fraction_reward("faucet", 1)

execute_plan()
```

Sample 2

```
import numpy as np

reset_reward() # Reset reward for a new task
set_joint_fraction_reward("faucet",
    1) # Turn on the faucet by setting the joint fraction to 1 (open)

execute_plan(2)
```

Code-as-Policies

Sample 1

```
reset()
faucet_handle_position = get_object_position("faucet_handle")
end_effector_to(faucet_handle_position)
end_effector_close()
normalized_joint_position = get_normalized_joint_position("faucet")
end_effector_open()
```

Sample 2

```
#python
import numpy as np

reset()
faucet_handle_pos = get_object_position("faucet_handle")
end_effector_to(faucet_handle_pos)
end_effector_open()
end_effector_close()

normalized_faucet_position = get_normalized_joint_position("faucet")
target_faucet_position = np.clip(normalized_faucet_position + 0.5, 0, 1)

end_effector_to([faucet_handle_pos[0],
    faucet_handle_pos[1], faucet_handle_pos[2] + 0.1])
end_effector_to([faucet_handle_pos[0],
    faucet_handle_pos[1], faucet_handle_pos[2]])
end_effector_open()
```

A.8 Full outputs for the interactive examples

Below we show the output reward code from our system for the two interactive examples: tasking a quadruped robot to perform moonwalk, and tasking a dexterous manipulator robot to place an apple in a drawer.

Quadruped moonwalk

Instruction 1: Make the robot stand upright on two back feet like a human.

```
import numpy as np

reset_reward()
set_torso_targets(0.65,
    np.deg2rad(90), np.deg2rad(0), (0.0, 0.0), None, 0, None)

set_feet_pos_parameters("front_left", 0.65, None, None)
set_feet_pos_parameters("back_left", 0.0, None, None)
set_feet_pos_parameters("front_right", 0.65, None, None)
set_feet_pos_parameters("back_right", 0.0, None, None)

execute_plan()
```

Instruction 2: Good, you actually don't need to keep the front paws at certain height, just leave it to the controller.

```
import numpy as np

reset_reward()
set_torso_targets(0.65,
    np.deg2rad(90), np.deg2rad(0), (0.0, 0.0), None, 0, None)

set_feet_pos_parameters("back_left", 0.0, None, None)
set_feet_pos_parameters("back_right", 0.0, None, None)

# Set stepping parameters for all feet to disable stepping motion
for foot in ["front_left", "rear_left", "front_right", "rear_right"]:
  set_feet_stepping_parameters(foot, 0.0, 0.0, 0.0, 0.0, 0.0, False)


execute_plan()
```

Instruction 3: Good, now make the robot do a moon-walk while stand up like this.

```
import numpy as np

reset_reward()
set_torso_targets(0.65, np.deg2rad(90), np.deg2rad(0), None, ( -0.2 , 0.0),
    0, None)

set_feet_pos_parameters("back_left", None, None, None)
set_feet_pos_parameters("back_right", None, None, None)

set_feet_stepping_parameters("back_left", 0.5, 0.5, 0.5, 0.1, -0.2, True)
set_feet_stepping_parameters("back_right", 0.5, 0.5, 0.0, 0.1, -0.2, True)
set_feet_stepping_parameters("front_left", 0.0, 0.0, 0.0, 0.0, 0.0, False)
set_feet_stepping_parameters("front_right", 0.0, 0.0, 0.0, 0.0, 0.0, False)

execute_plan()
```

Instruction 4: Moon walk means the robot should walk backward while the feet swings as if they are moving forward. Correct your answer and also make it walk at a speed of 0.5m/s.

```
import numpy as np

reset_reward()
set_torso_targets(0.65, np.deg2rad(90), np.deg2rad(0), None, ( -0.5 , 0.0),
    0, None)

set_feet_pos_parameters("back_left", None, None, None)
set_feet_pos_parameters("back_right", None, None, None)

set_feet_stepping_parameters("back_left", 0.5, 0.5, 0.5, 0.1,  0.2 , True)
set_feet_stepping_parameters("back_right", 0.5, 0.5, 0.0, 0.1,  0.2 , True)
set_feet_stepping_parameters("front_left", 0.0, 0.0, 0.0, 0.0, 0.0, False)
set_feet_stepping_parameters("front_right", 0.0, 0.0, 0.0, 0.0, 0.0, False)

execute_plan()
```

Dexterous Manipulator place an apple in a drawer

Instruction 1: Open the drawer.

```
import numpy as np

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_l2_distance_reward("palm", "drawer_handle")
set_joint_fraction_reward("drawer", 1) # 1 means open

execute_plan()
```

Instruction 2: Good, now put the apple inside the drawer while keep it open. Assume drawer is on the floor so no need to lift up the apple.

```
import numpy as np

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_l2_distance_reward("palm", "apple")
set_l2_distance_reward("apple", "drawer_center")
set_joint_fraction_reward("drawer", 1)

execute_plan()
```

Instruction 3: Good, now release the apple and move hand away.

```
import numpy as np

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_l2_distance_reward("palm", "rest_position")

execute_plan(4)
```

Instruction 4: Now close the drawer.

```
import numpy as np

reset_reward()
    # This is a new task so reset reward; otherwise we don't need it
set_l2_distance_reward("palm", "drawer_handle")
set_joint_fraction_reward("drawer", 0)

execute_plan()
```

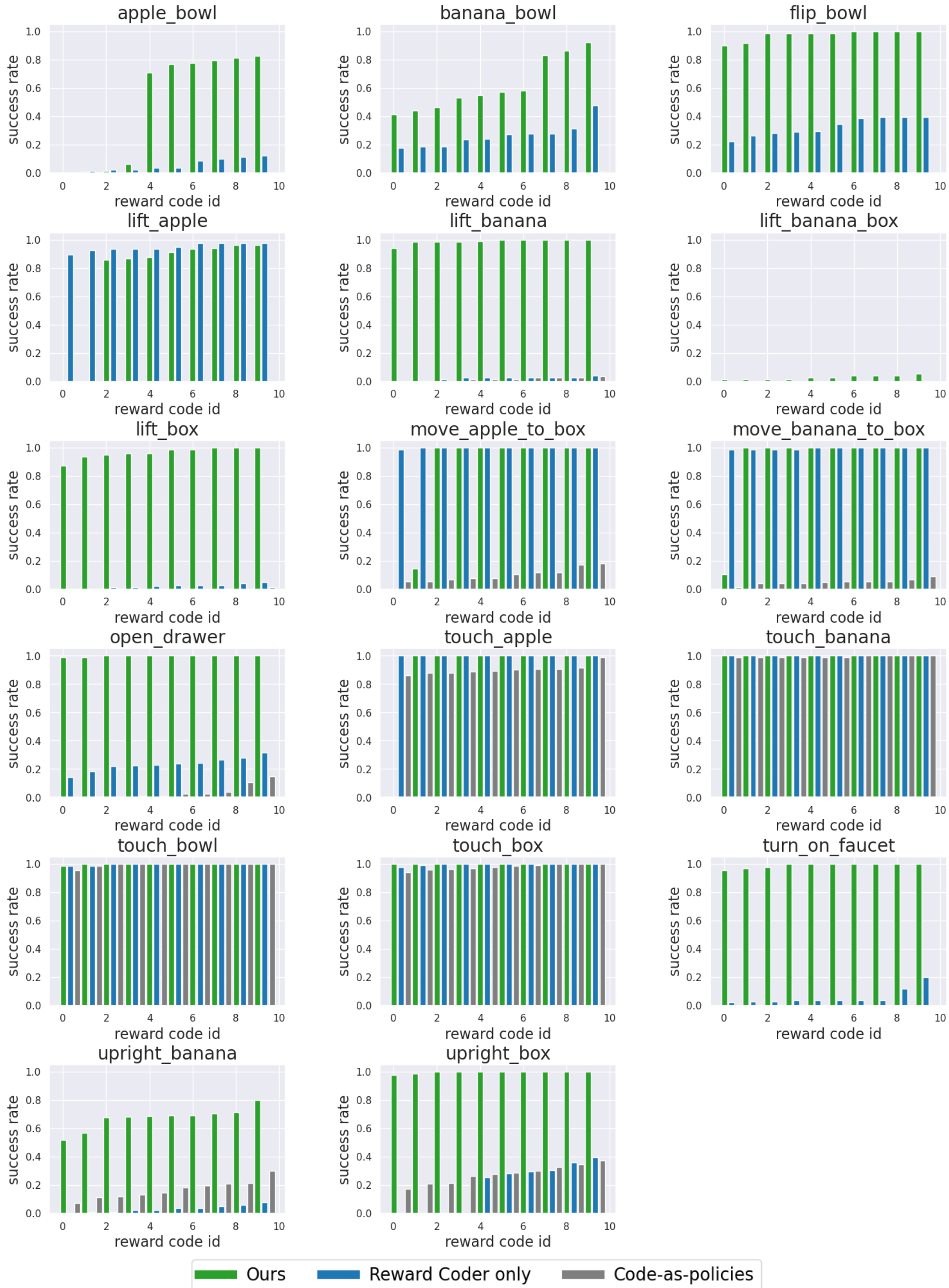

Figure 8: Full evaluation results for the Dexterous Manipulator robot. Note that to better interpret the results, we order the generated reward code in the figures based on the mean success rate.

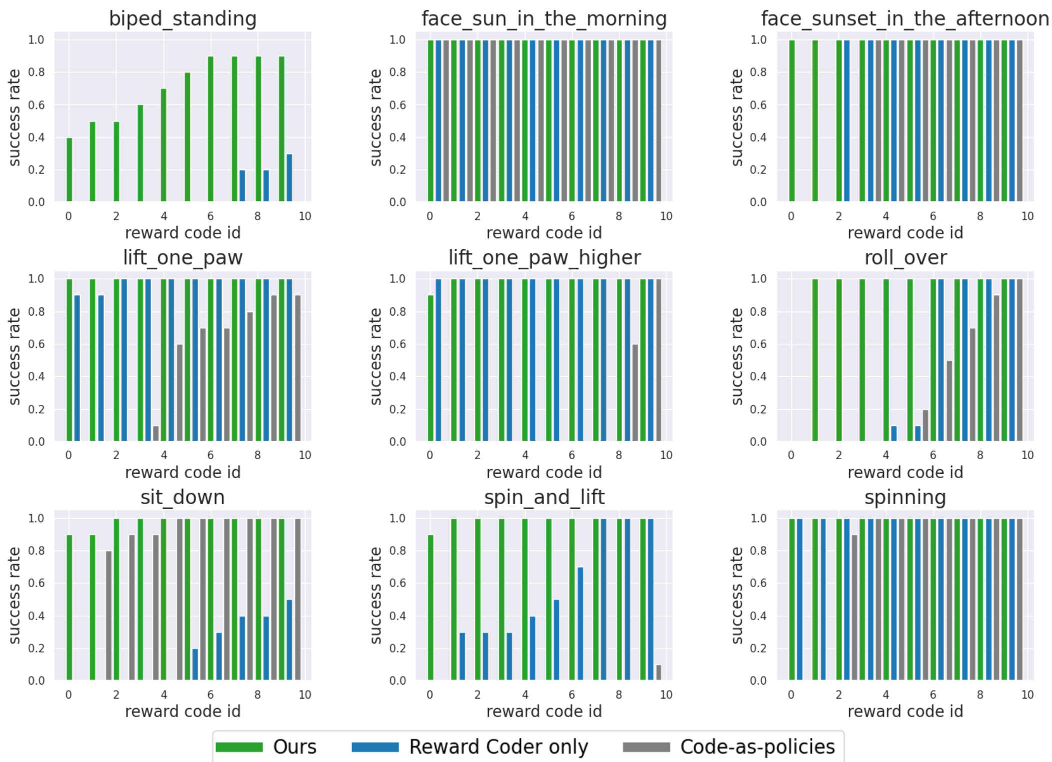

Figure 9: Full evaluation results for the Quadruped robot. Note that to better interpret the results, we order the generated reward code in the figures based on the mean success rate.